FIG. 1

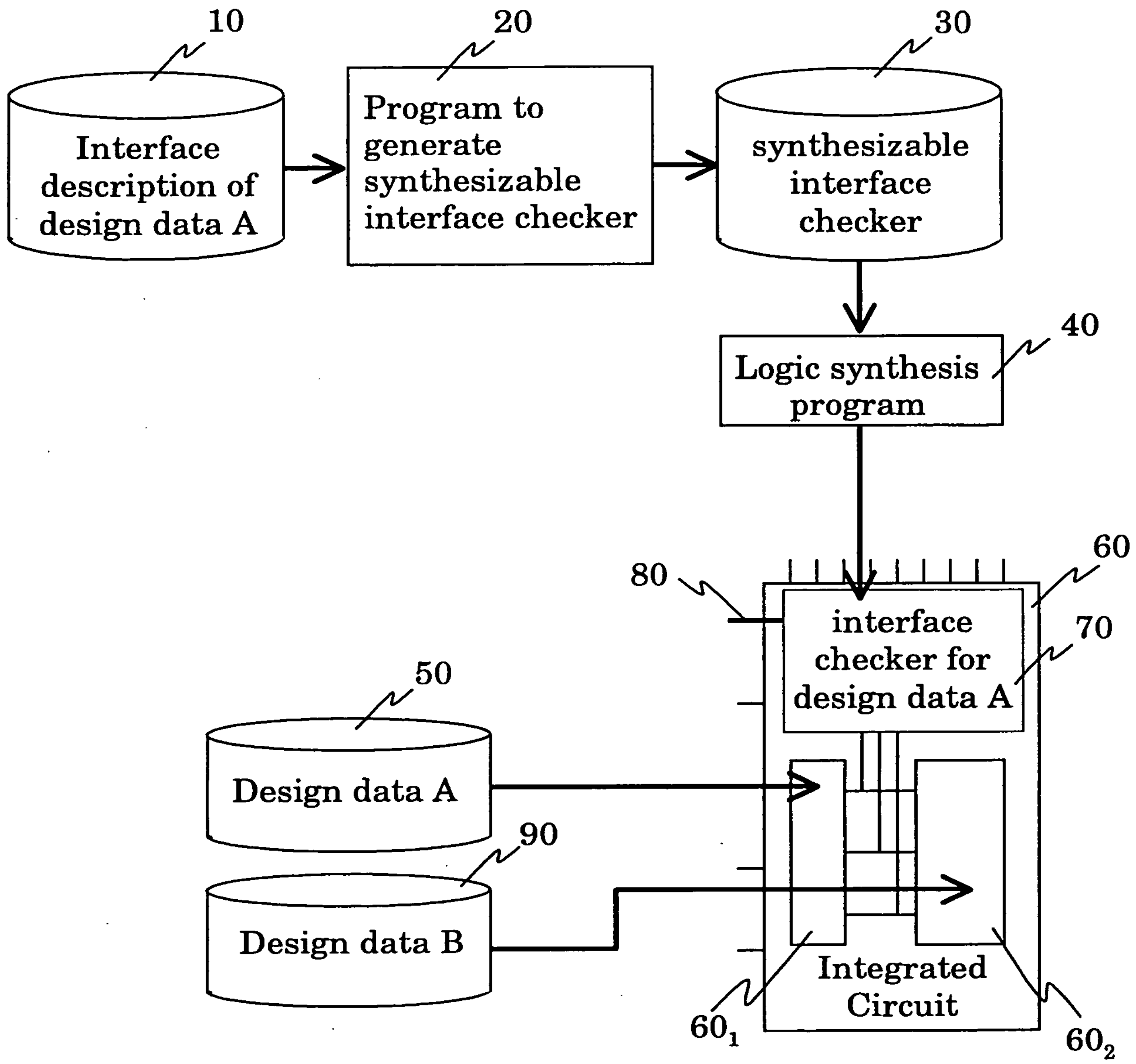
10
Interface description of design data A
20
Program to generate synthesizable interface checker
30
synthesizable interface checker
40
Logic synthesis program
60
80
interface checker for design data A
70
50
Design data A
90
Design data B
Integrated Circuit
60_1
60_2

FIG. 2

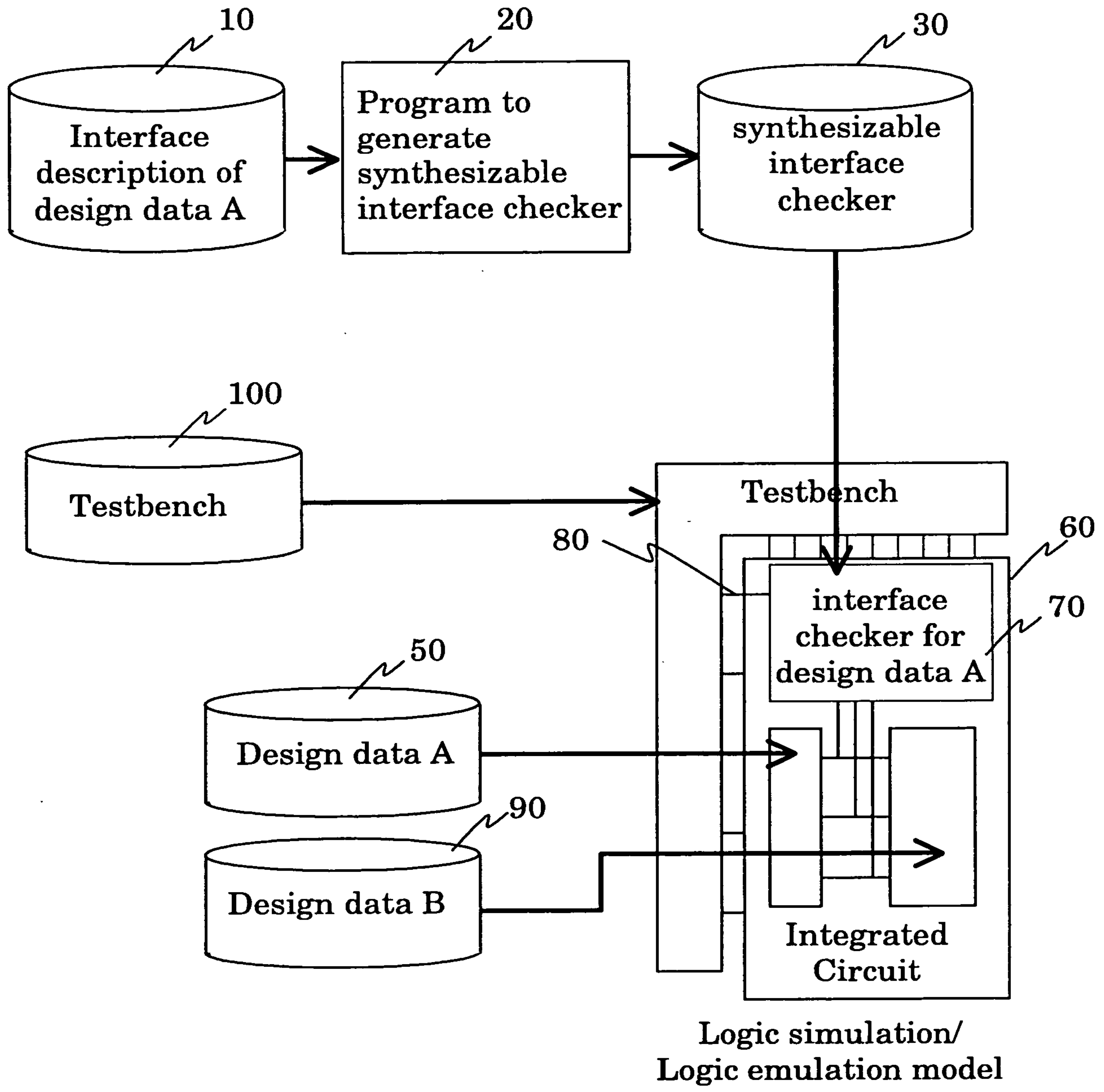
10
Interface description of design data A
20
Program to generate synthesizable interface checker
30
synthesizable interface checker
100
Testbench
Testbench
80
60
interface checker for design data A
70
50
Design data A
90
Design data B
Integrated Circuit
Logic simulation/ Logic emulation model

FIG. 3

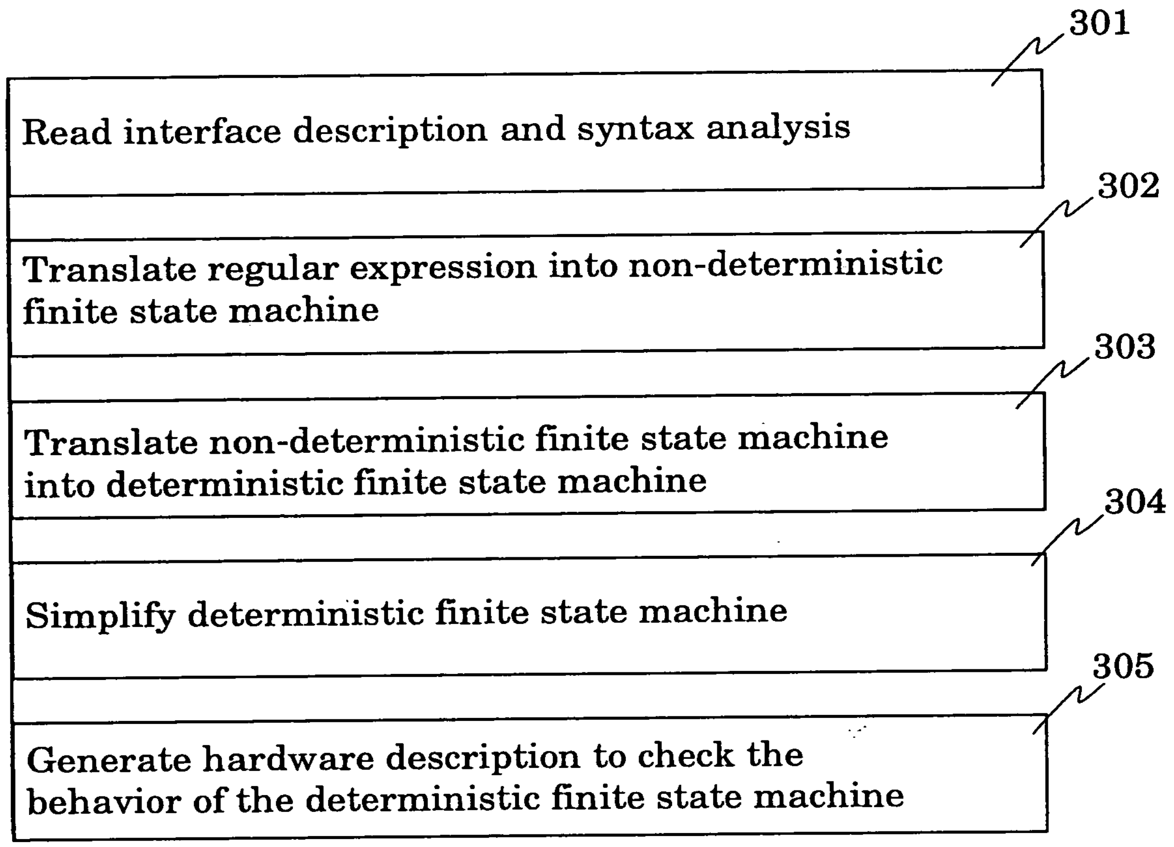
301
Read interface description and syntax analysis
302
Translate regular expression into non-deterministic finite state machine
303
Translate non-deterministic finite state machine into deterministic finite state machine
304
Simplify deterministic finite state machine
305
Generate hardware description to check the behavior of the deterministic finite state machine

FIG. 4

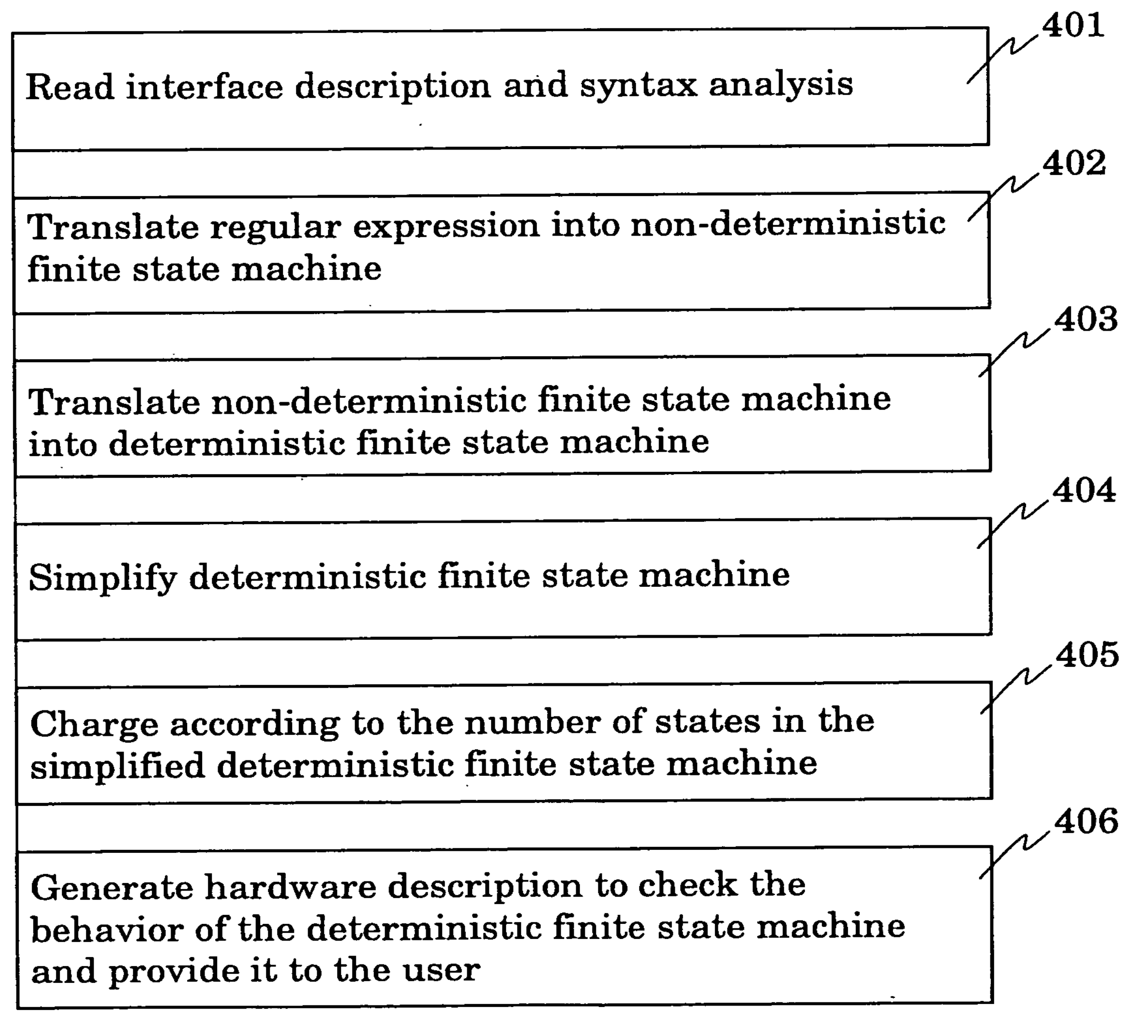
401
Read interface description and syntax analysis
402
Translate regular expression into non-deterministic finite state machine
403
Translate non-deterministic finite state machine into deterministic finite state machine
404
Simplify deterministic finite state machine
405
Charge according to the number of states in the simplified deterministic finite state machine
406
Generate hardware description to check the behavior of the deterministic finite state machine and provide it to the user

FIG. 5

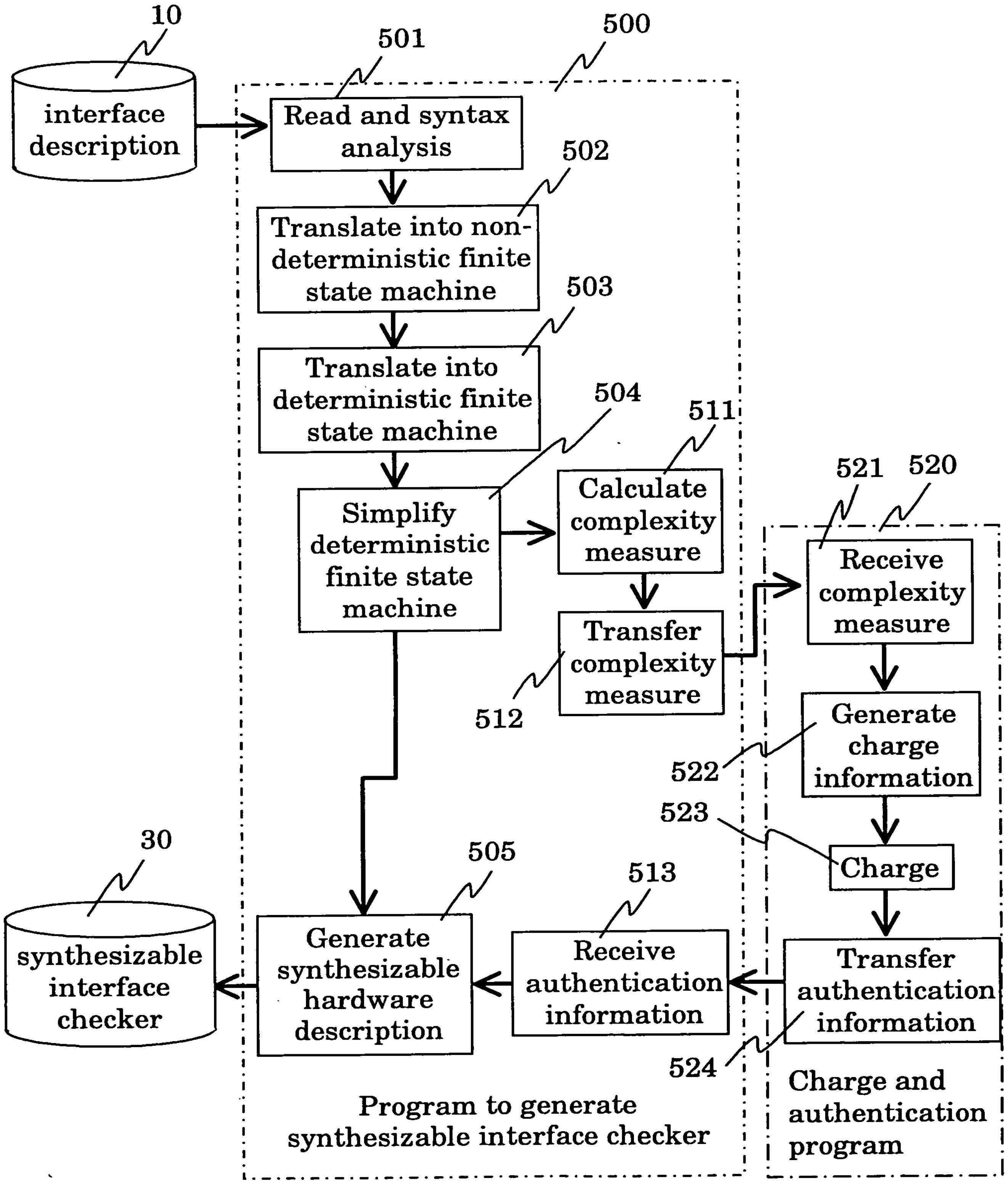
10
interface description
500
501
Read and syntax analysis
502
Translate into non-deterministic finite state machine
503
Translate into deterministic finite state machine
504
Simplify deterministic finite state machine
511
Calculate complexity measure
512
Transfer complexity measure
521
520
Receive complexity measure
Generate charge information
522
523
Charge
Transfer authentication information
524
513
Receive authentication information
505
Generate synthesizable hardware description
30
synthesizable interface checker
Program to generate synthesizable interface checker
Charge and authentication program

FIG. 6

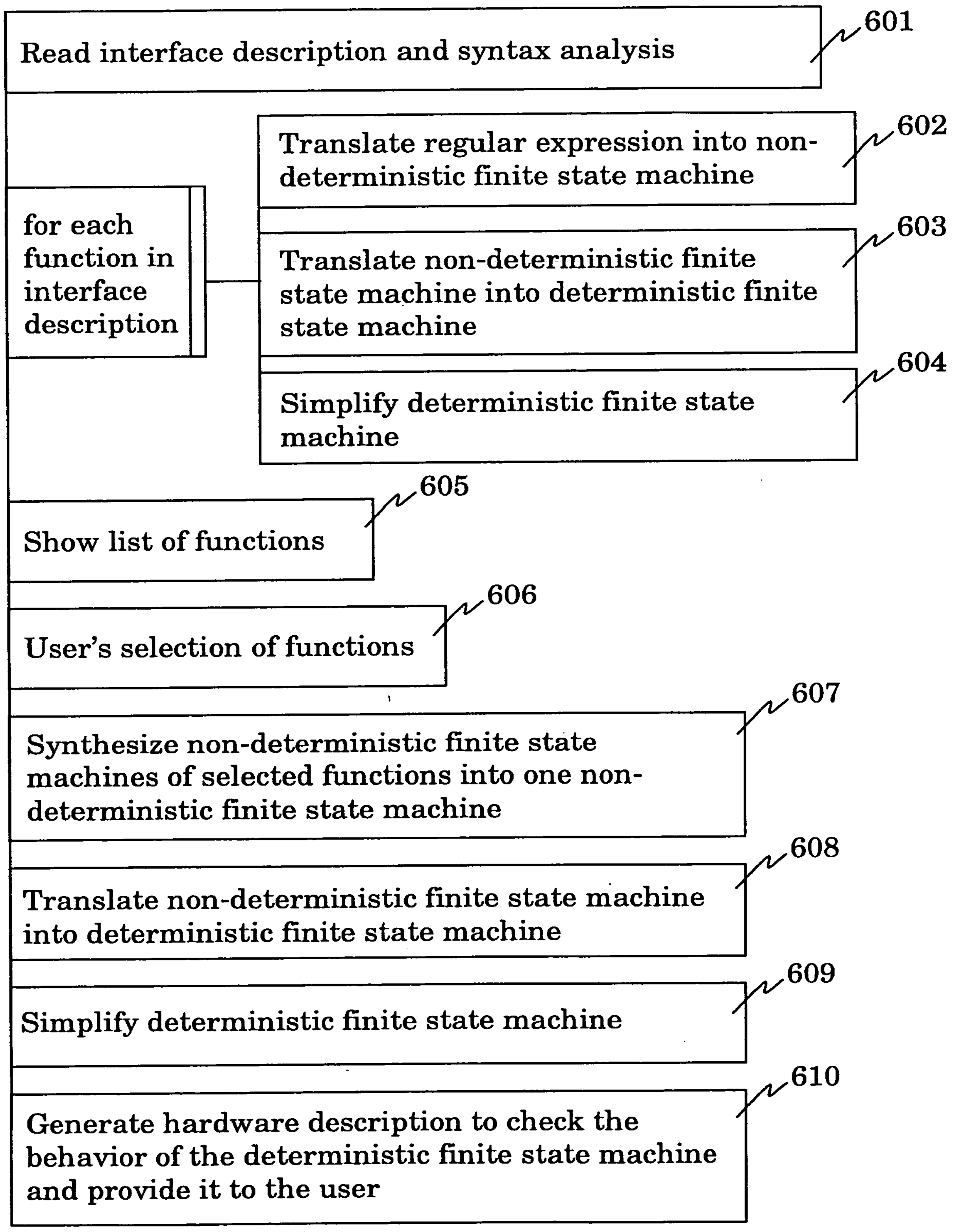
Read interface description and syntax analysis
601
for each function in interface description
Translate regular expression into non-deterministic finite state machine
602
Translate non-deterministic finite state machine into deterministic finite state machine
603
Simplify deterministic finite state machine
604
Show list of functions
605
User's selection of functions
606
Synthesize non-deterministic finite state machines of selected functions into one non-deterministic finite state machine
607
Translate non-deterministic finite state machine into deterministic finite state machine
608
Simplify deterministic finite state machine
609
Generate hardware description to check the behavior of the deterministic finite state machine and provide it to the user
610

FIG. 7

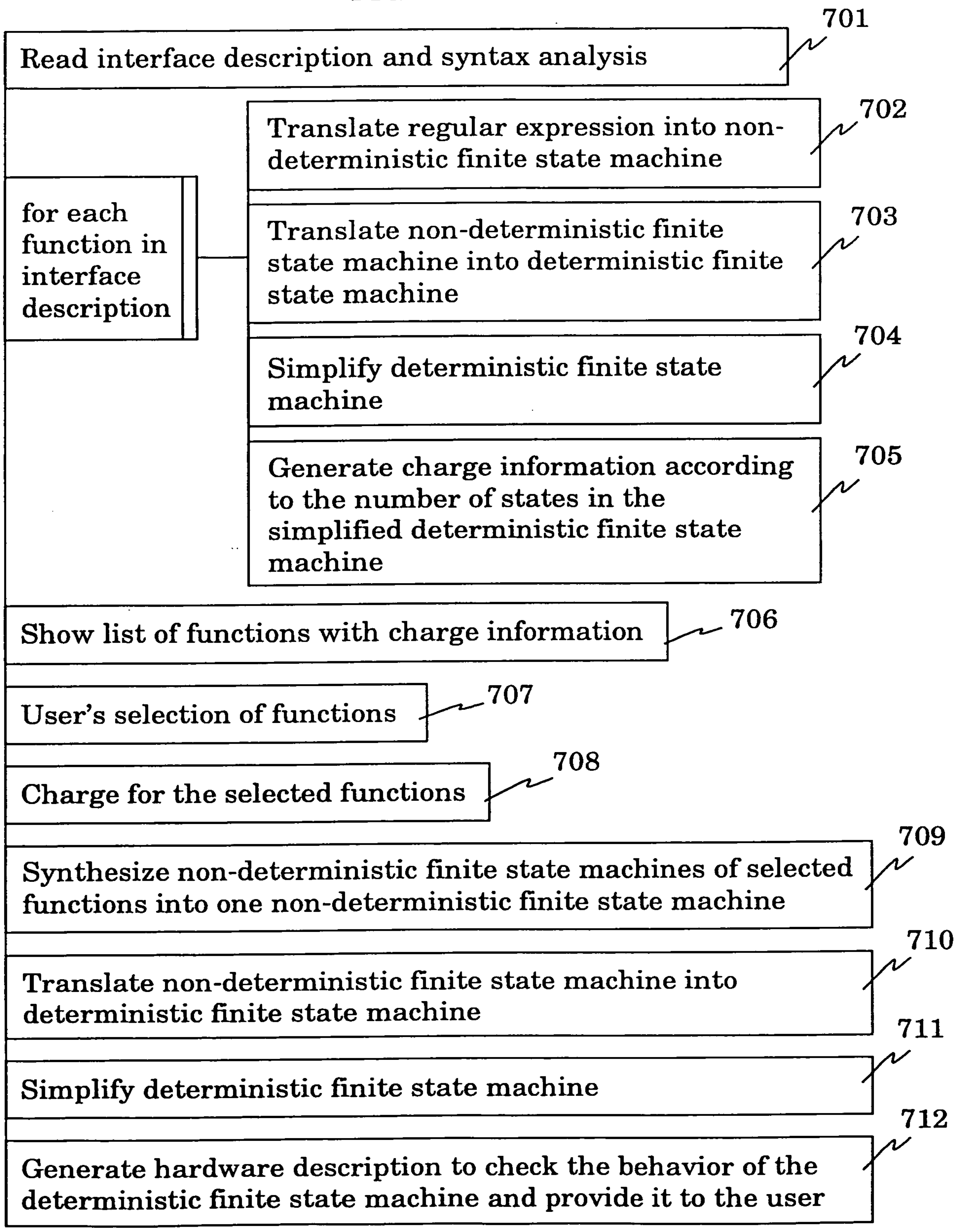
Read interface description and syntax analysis
701
for each function in interface description
Translate regular expression into non-deterministic finite state machine
702
Translate non-deterministic finite state machine into deterministic finite state machine
703
Simplify deterministic finite state machine
704
Generate charge information according to the number of states in the simplified deterministic finite state machine
705
Show list of functions with charge information
706
User's selection of functions
707
Charge for the selected functions
708
Synthesize non-deterministic finite state machines of selected functions into one non-deterministic finite state machine
709
Translate non-deterministic finite state machine into deterministic finite state machine
710
Simplify deterministic finite state machine
711
Generate hardware description to check the behavior of the deterministic finite state machine and provide it to the user
712

FIG. 8

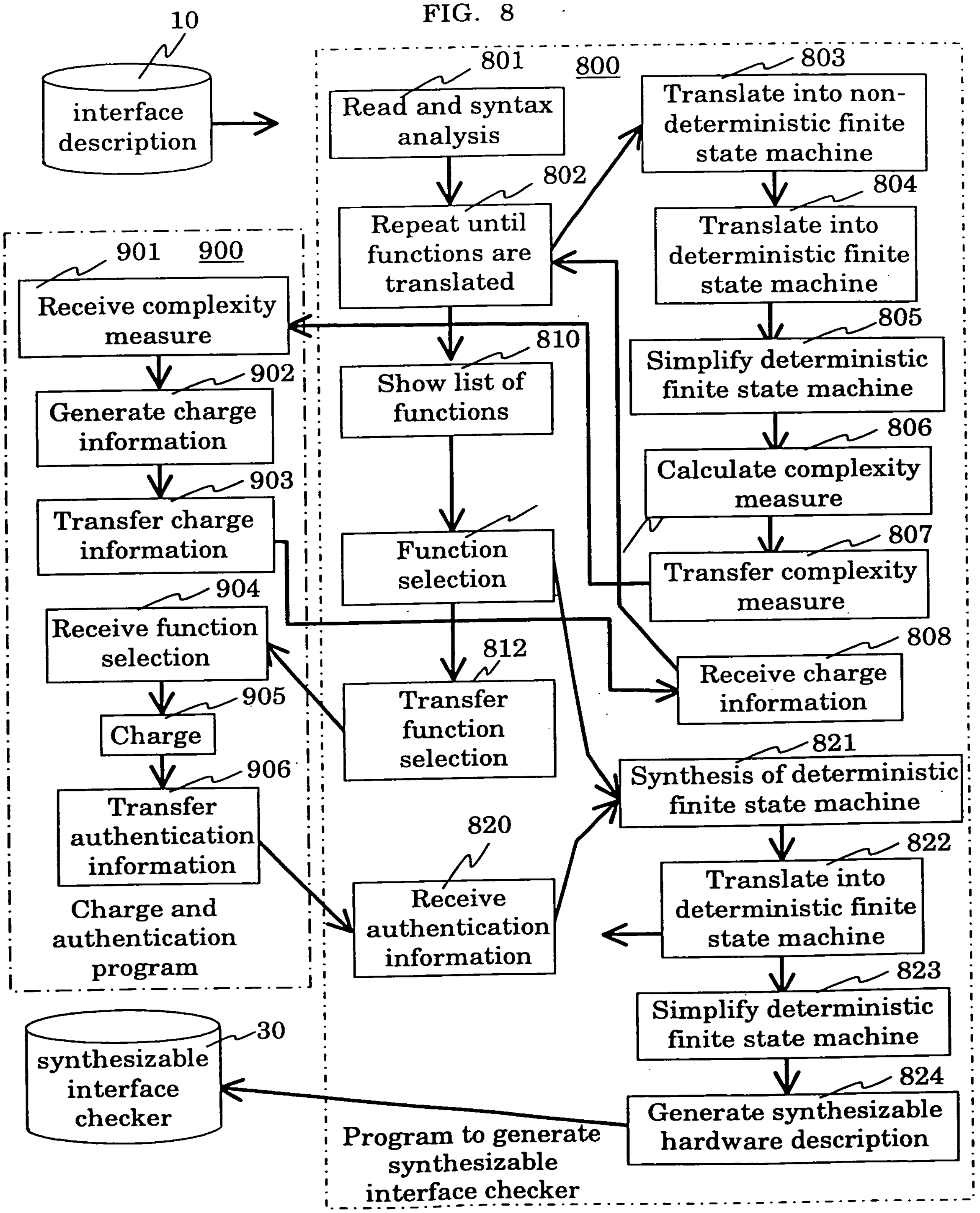
10
interface description
800
Program to generate synthesizable interface checker
801
Read and syntax analysis
802
Repeat until functions are translated
803
Translate into non-deterministic finite state machine
804
Translate into deterministic finite state machine
805
Simplify deterministic finite state machine
806
Calculate complexity measure
807
Transfer complexity measure
808
Receive charge information
810
Show list of functions
Function selection
812
Transfer function selection
820
Receive authentication information
821
Synthesis of deterministic finite state machine
822
Translate into deterministic finite state machine
823
Simplify deterministic finite state machine
824
Generate synthesizable hardware description
900
Charge and authentication program
901
Receive complexity measure
902
Generate charge information
903
Transfer charge information
904
Receive function selection
905
Charge
906
Transfer authentication information
30
synthesizable interface checker

FIG. 9

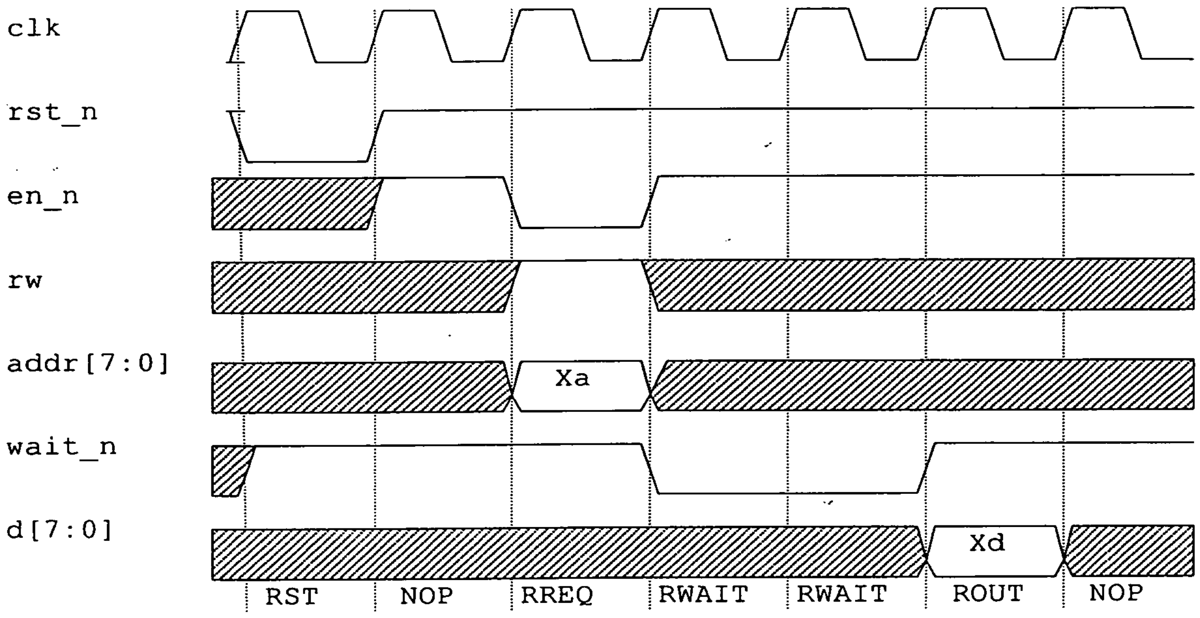
clk
rst_n
en_n
rw
addr[7:0]
Xa
wait_n
d[7:0]
Xd
RST
NOP
RREQ
RWAIT
RWAIT
ROUT
NOP

FIG. 10

```
define interface simple_memory;
define port;
  input.clock  clk;
  input.control  rst_n;
  input.control  rw;
  input.control  en_n;
  input.data [7:0]  addr;
  output.control  wait_n;
  inout.data [7:0]  d;
endport

define alphabet;
  signal set={ clk, rst_n, en_n, rw, addr, wait_n, d };
  NOP:     { posedge, 1, 1, *, *, 1, Z };
  RST:     { posedge, 0, *, *, *, *, Z };
  RREQ(Xa):{ posedge, 1, 0, 1, $i.Xa, 1, Z };
  RWAIT:   { posedge, 1, 1, *, *, 0, Z };
  ROUT(Xd):{ posedge, 1, 1, *, *, 1, $o.Xd };
endalphabet

define word;
  void nop : NOP NOP*;
  void reset : RST RST*;
  data(Xd) byte_read(Xa):RREQ(Xa) RWAIT RWAIT ROUT(Xd);
endword

define sentence;
  reset [ reset | nop | byte_read(Xa) ]*;
endsentence
endinterface
```

FIG. 11

Edge without name is ε transition

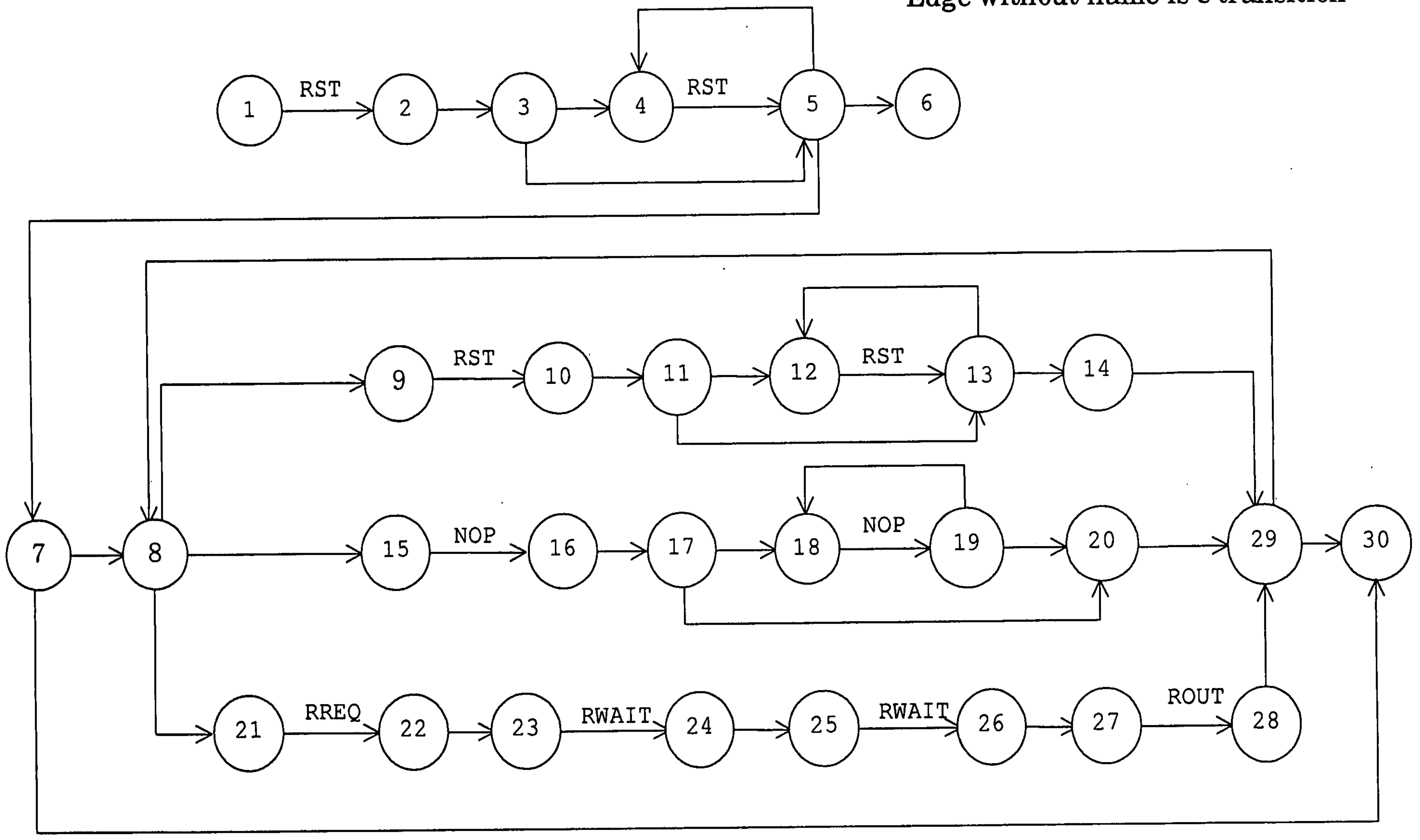

FIG. 12

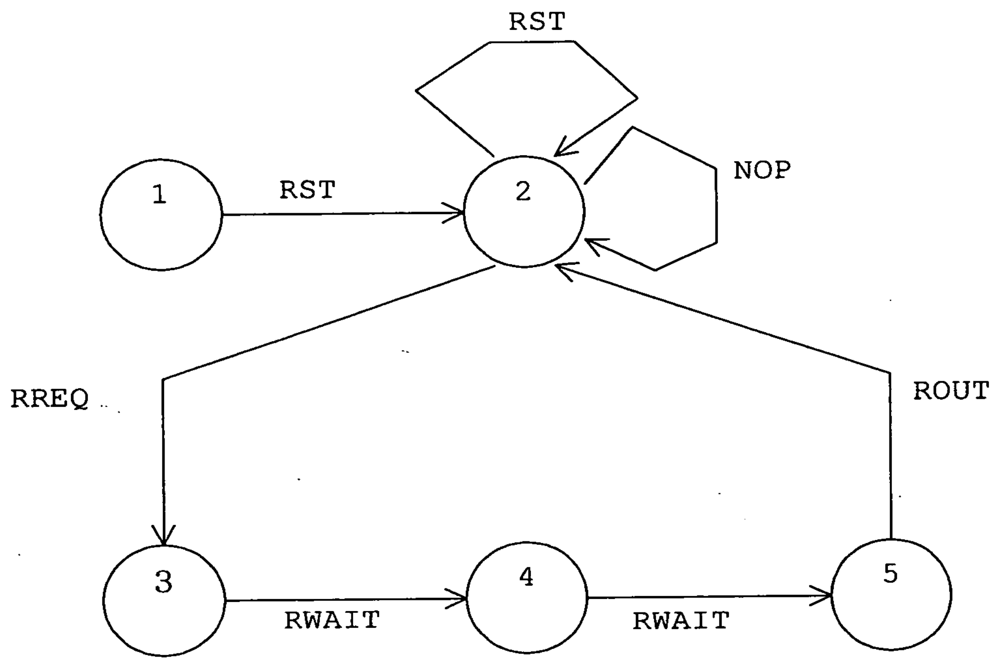

FIG. 13

```
`define NOP(nstate) if ((clk == 1)&&(rst_n == 1)&&(en_n == 1)&&(wait_n == 1))\
         begin \
           state <= nstate; \
         end else
`define RST(nstate) if ((clk == 1)&&(rst_n == 0))\
         begin \
           state <= nstate; \
         end else
`define RREQ(nstate) if ((clk == 1)&&(rst_n == 1)&&(en_n == 0)&&(rw == 1)\
         && (wait_n == 1)) \
         begin \
           state <= nstate;\
         end else
`define RWAIT(nstate) if ((clk == 1)&&(rst_n == 1)&&(en_n == 1)&&(wait_n == 0))\
         begin \
           state <= nstate; \
         end else
`define ROUT(nstate) if ((clk == 1)&&(rst_n == 1)&&(en_n == 1)&&(wait_n == 1)) \
         begin \
           state <= nstate; \
         end else
`define REJECT begin reject <= 1'b1; state <= `reject_state; end

`define initial_state 3'h0
`define reject_state 3'h7
`define s1 3'h1
`define s2 3'h2
`define s3 3'h3
`define s4 3'h4
`define s5 3'h5
```

FIG. 14

```
module simple_memory( clk, rst_n, rw, en_n,
addr, wait_n, d, reject );
input clk, rst_n, rw, en_n, addr, wait_n, d;
wire clk;
wire rst_n;
wire rw;
wire en_n;
wire [7:0] addr;
wire wait_n;
wire [7:0] d;
reg reject;
reg [2:0] state;
always @(posedge clk) begin
  case(state)
  `initial_state: begin
    reject <= 1'b0;
    `RST(`s2)
    begin end
  end
  `s1: begin
    `RST(`s2)
    `REJECT
  end
  `s2: begin
    `RST(`s2)
    `NOP(`s2)
    `RREQ(`s3)
    `REJECT
  end
  `s3: begin
    `RWAIT(`s4)
    `REJECT
  end
  `s4: begin
    `RWAIT(`s5)
    `REJECT
  end
  `s5: begin
    `ROUT(`s2)
    `REJECT
  end
  default: begin
    state <= `initial_state;
    reject <= 1'b0;
  end
end
endmodule
```